

**1) (2 ptos.)** Suponga una estructura de datos que tiene un invariante I1 que permite que todas sus operaciones se realicen eficientemente. Suponga que existe otro invariante I2 (más fuerte que I1) que es computacionalmente costoso de lograr pero permite que algunas operaciones se realicen más eficientemente (por ejemplo, podría eliminar repeticiones, ordenar alfabéticamente, etc.). Llamemos Optimizar() a la función privada y costosa que restablece el invariante I2.

- Escriba su pre y su post condición.
- ¿De qué manera modificaría el módulo para valerse de Optimizar() e I2? Considere algoritmos, interfaz, etc.
- Luego del punto anterior, ¿cuál sería la signatura de la función de abstracción?
- Luego de b), ¿cómo quedaría expresada la complejidad de las operaciones en la signatura?

Justifique todas sus respuestas. No es necesario que repita definiciones.

**2) (2 ptos.)** Para cada uno de los algoritmos de ordenamiento siguientes, completar la siguiente información

- Complejidad temporal en el Peor Caso,
- Complejidad temporal en el Caso promedio
- Memoria adicional requerida
- En algunos contextos es útil suspender un algoritmo de ordenamiento, agregar nuevos elementos al arreglo y luego continuarlo. Algunos algoritmos soportan esta noción sin modificaciones, otros requieren algún tipo de modificación para permitirlo y otros son sencillamente incompatibles con el concepto. Clasifique los siguientes algoritmos en apto sin modificaciones, apto con modificaciones y no aptos.

En cada uno de los casos se requiere una justificación (entendiéndose como tal aclaraciones, consideraciones necesarias, suposiciones o explicaciones para que valga lo referido en el campo correspondiente). Ninguna justificación debería tener más de 3 o 4 renglones. No dejar ningún ítem sin responder. Eventualmente poner "?" y justificar.

Algoritmos: Selection Sort, Insertion Sort, Quick Sort, Merge Sort, y Heapsort.

**3) (2 ptos.)** Describir (seudocódigo o explicación clara) de las operaciones de Búsqueda, Inserción y Borrado correspondientes al Hashing Extensible o Dinámico (memoria secundaria) vistos en clase. Explicar cualquier decisión de diseño tomada y analizar la complejidad de las operaciones.

**4) (2 ptos)** Escribir el pseudocódigo de un algoritmo que, dado un árbol binario con información en los nodos, determine si es un ABB Balanceado (de acuerdo a la definición de balanceamiento de los árboles AVL). Analizar la complejidad del algoritmo propuesto. A partir de la respuesta anterior, ¿bajo alguna circunstancia o condición podría tener sentido utilizar este algoritmo para mejorar la eficiencia de una implementación de diccionario? Justificar.

5) (2 ptos) Dada la siguiente especificación, indicar si tiene errores, cuáles y por qué son errores, y cómo los arreglaría (puede axiomatizar o explicar los eventuales arreglos).

TAD Persona

generadores

crear\_persona: dni x edad  $\rightarrow$  persona

observadores

= persona x persona  $\rightarrow$  bool

FIN

TAD Regalo es Nat

TAD Fiesta

generadores

invitar: conj(personas) x fecha  $\rightarrow$  fiesta

llega\_invitado: persona x regalo x fiesta  $\rightarrow$  fiesta

observadores

invitados: fiesta  $\rightarrow$  conj(personas)

que\_regalo\_trajo?: fiesta f x persona p  $\rightarrow$  regalo ( $p \in$  invitados(f))

suma\_edades: fiesta  $\rightarrow$  edad

axiomas

suma\_edades(invitar(c)) = 0

suma\_edades(llega\_invitado(crear\_persona(d, e), r, f)) = e + suma\_edades(f)

que\_regalo\_trajo?(invitar(c), p) = 0

que\_regalo\_trajo?(llega\_invitado(p, r, f), p') = if  $p=p'$  then r else que\_regalo\_trajo?(f, p') fi

invitados(invitar(c), f) = c

invitados(llega\_invitado(p, r, f)) = invitados(f)

I